Combiner connaissances expertes, hors-ligne, transientes et en ligne pour l'exploration Monte-Carlo

Louis Chatriot, Christophe Fiter, Guillaume Chaslot, Sylvain Gelly, Jean-Baptiste Hoock, J. Perez, Arpad Rimmel, Olivier Teytaud

Combiner connaissances expertes, hors-ligne, transientes et en ligne pour l'exploration Monte-Carlo

Apprentissage et MC

Louis Chatriot— Christophe Fiter— Guillaume Chaslot— Sylvain Gelly— Jean-Baptiste Hoock— Julien Perez— Arpad Rimmel— Olivier Teytaud

TAO (Inria), LRI, UMR 8623 (CNRS - Univ. Paris-Sud),
bat 490 Univ. Paris-Sud 91405 Orsay, France, teytaud@lri.fr

RÉSUMÉ. *Nous combinons pour de l'exploration Monte-Carlo d'arbres de l'apprentissage artificiel à 4 échelles de temps :*

– regret en ligne, via l'utilisation d'algorithmes de bandit et d'estimateurs Monte-Carlo ;

– de l'apprentissage transient, via l'utilisation d'estimateur rapide de Q-fonction (RAVE, pour Rapid Action Value Estimate) qui sont appris en ligne et utilisés pour accélérer l'exploration mais sont ensuite peu à peu laissés de côté à mesure que des informations plus fines sont disponibles ;

– apprentissage hors-ligne, par fouille de données de jeux ;

– utilisation de connaissances expertes comme information a priori.

L'algorithme obtenu est plus fort que chaque élément séparément. Nous mettons en évidence par ailleurs un dilemne exploration-exploitation dans l'exploration Monte-Carlo d'arbres et obtenons une très forte amélioration par calage des paramètres correspondant.

ABSTRACT. *We combine for Monte-Carlo exploration machine learning at four different time scales:*

– online regret, through the use of bandit algorithms and Monte-Carlo estimates;

– transient learning, through the use of rapid action value estimates (RAVE) which are learnt online and used for accelerating the exploration and are thereafter neglected;

– offline learning, by data mining of datasets of games;

– use of expert knowledge coming from the old ages as prior information.

The resulting algorithm is stronger than each element separately. We finally emphasize the exploration-exploitation dilemna in the Monte-Carlo simulations and show great improvements that can be reached with a fine tuning of related constants.

MOTS-CLÉS : *Exploration d'Arbres par Monte-Carlo, Apprentissage en ligne, Apprentissage hors-ligne, Apprentissage transient, Connaissances expertes, Combinaison de prédicteurs*

KEYWORDS: *Monte-Carlo Tree-Search, Online Learning, Offline Learning, Transient Learning, Expert knowledge, Combining predictors*

1. Introduction

Les diverses définitions des termes du jeu de go utilisées dans cet article peuvent être trouvées sur le site internet `http://senseis.xmp.net`

Le Monte-Carlo Tree Search, MCTS (Kocsis *et al.*, 2006; Coulom, 2006; Chaslot *et al.*, 2006), est un outil récent pour les tâches de planification difficiles. D'impressionnants résultats ont déjà été publiés (Coulom, 2007; Gelly *et al.*, 2007). Dans ce papier, nous présentons l'introduction de connaissance d'experts et d'apprentissages artificiels sur plusieurs échelles de temps différentes pour un problème spécifique de prise de décision, le jeu de Go.

L'algorithme MCTS (Chaslot *et al.*, 2007; Coulom, 2006) consiste à construire un arbre dans lequel chaque noeud est une situation de l'environnement considéré et les branches sont des actions pouvant être prises par l'agent. Le point important du MCTS est que l'arbre est fortement déséquilibré : un biais est appliqué en faveur des parties importantes de l'arbre. Ainsi, l'intérêt est mis aux zones de l'arbre ayant une plus forte espérance de gain. Plusieurs algorithmes ont été proposés : UCT(Kocsis *et al.*, 2006) (Upper Confidence Trees, i.e. UCB applied to Trees), qui porte son attention sur les noeuds de l'arbre ayant le plus grand nombre de simulations gagnantes *plus* un terme favorisant les noeuds peu explorés, AMAF (All Moves As First, (Gelly *et al.*, 2007)), qui fait un compromis entre le score à la UCB1 et des informations heuristiques obtenues par permutations de simulations, et BAST (Coquelin *et al.*, 2007) (Bandit Algorithm for Search in Trees), qui se focalise quant à lui sur le nombre total de noeuds dans l'arbre. D'autres algorithmes proches sont également proposés dans (Chaslot *et al.*, 2007; Coulom, 2007), combinant MCTS et des statistiques de parties de joueurs professionnels.

Dans le contexte du jeu de Go, les noeuds de l'arbre incorporent la configuration du tableau de jeu qu'ils représentent (aussi appelée coloration du plateau), et des statistiques telles que le nombre de victoires obtenues et le nombre total de parties jouées traversant ce noeud. Comme expliqué dans l'Algorithme 1, MCTS utilise ces statistiques afin d'étendre itérativement l'arbre des possibles dans la région où la plus forte espérance de gain aura été calculée. Après chaque simulation à partir de la racine de l'arbre, les statistiques de gain et de perte sont mises à jour dans tous les noeuds ayant été parcourus et un nouveau noeud est créé dans l'arbre ; ce nouveau noeud correspond au premier noeud visité lors de la partie simulation Monte-Carlo (i.e. la première situation rencontrée dans la simulation et qui n'était pas encore dans l'arbre).

Ainsi, afin de concevoir des simulations à partir de la racine de l'arbre (la position courante) jusqu'à la position finale (victoire ou défaite), plusieurs décisions doivent être prises : (i) dans l'arbre, jusqu'à parvenir à une feuille (ii) hors de l'arbre, jusqu'à ce que la partie soit terminée. Les buts de ces deux choix de transition sont différents. Dans l'arbre, la transition doit assurer un bon compromis entre exploration et exploitation. Hors de l'arbre, les simulations, habituellement nommées simulation Monte-Carlo, doivent permettre d'obtenir une bonne estimation de la probabilité de victoire.

L'Algorithme 1 résume le planning par méthode Monte-Carlo.

Algorithm 1 Pseudo-code d'un algorithme MCTS appliqué à un jeu à somme nulle (comme le Go ou les échecs). T est un arbre de positions, avec chaque noeud contenant des statistiques (nombre de parties gagnées et perdues dans les simulations passant par ce noeud). En ce qui concerne la décision à la toute fin du pseudo-code, choisir le coup le plus simulé est la solution la plus sure ; d'autres solutions comme le choix du meilleur "ratio nombre de parties gagnées sur nombre de parties total" ne sont pas robustes car il peut y avoir très peu de simulations. Ici la récompense pour chaque partie est binaire (gagnée ou perdue) mais des distributions arbitraires peuvent être utilisées.

Initialiser T à un noeud unique, représentant l'état courant.
while Temps restant > 0 **do**
  Simuler une partie jusqu'à une feuille (une position) L de T (à l'aide de l'algorithme de bandit, cf Algorithme 3).
  Choisir un fils (un successeur) L' de L.
  Simuler une partie depuis la position L' jusqu'à la fin de la partie (cf Algorithme 2).
  Développer l'arbre : ajouter L' en tant que fils de L dans T.
  Mettre à jour les statistiques dans tout l'arbre. Dans UCT, chaque noeud connaît combien de simulations gagnées (depuis ce noeud) ont été jouées ainsi que le nombre de simulations totales. Pour d'autres formes d'exploration d'arbre, il peut y avoir besoin de plus d'informations (information heuristique dans le cas d'AMAF, nombre total de noeuds dans le cas de BAST (Coquelin *et al.*, 2007)).
end while
Retourner le coup qui a été simulé le plus souvent à partir de la racine.

La fonction utilisée pour prendre les décisions hors de l'arbre est définie dans l'algorithme 2. Un atari se produit lorsqu'une chaîne (un groupe de pierres) peut être capturé en un coup. Des connaissances de Go ont été ajoutées dans cette partie, sous la forme de patterns 3x3 conçus pour jouer des parties plus réalistes.

La fonction utilisée pour choisir les coups dans l'arbre est présentée dans l'algorithme 3. Cette fonction est l'élément principal de notre programme : elle décide de la direction dans laquelle l'arbre sera étendu. Il existe plusieurs formules différentes ; nous présentons ici quelques versions (voir (Lai *et al.*, 1985),(Auer *et al.*, 2002),(Gelly *et al.*, 2007) pour UCB1, UCB-Tuned et AMAF respectivement) ; d'autres variantes très importantes (pour le cas de domaines infinis, d'un très grand nombre de bras, de suppositions spécifiques) peuvent être trouvées dans (Banks *et al.*, 1992),(Agrawal, 1995),(Dani *et al.*, 2006),(Coquelin *et al.*, 2007),(Chaslot *et al.*, 2007). Toutes sont basées sur l'idée de compromis entre exploitation (simuler dans le sens des coups à succès) et l'exploration (simuler les coups qui ne l'ont pas encore été ou peu).

L'algorithme AMAF a été dérivé de (Gelly *et al.*, 2007). Les simulations d'AMAF sont créées par permutation de coup dans la simulation réelle. Un progrès important,

Algorithm 2 Algorithme permettant de choisir un coup dans une simulation MC dans le cas du jeu de Go.

```
if le dernier coup est un atari then
  Sauver les pierres en atari.
else
  if il y a un emplacement vide parmi les 8 emplacements autour du dernier coup
  qui correspond à un pattern then
    Jouer au hasard sur l'un de ces emplacements (sortir).
  else
    if il y a un coup qui capture des pierres then
      Capturer les pierres. (fin de fonction)
    else
      if il reste des coups légaux then
        Jouer au hasard un coup légal. (fin de fonction)
      else
        Renvoyer "passe".
      end if
    end if
  end if
end if
```

le "Progressive Widening", de l'algorithme 3 consiste à considérer uniquement les $K(n)$ meilleurs coups (selon une heuristique choisie) à la n-ième simulation d'un noeud donné. Avec $K(n)$ une application non-décroissante de $\mathbb{N}$ dans $\mathbb{N}$.

Malheureusement, plusieurs problèmes surviennent : premièrement, la partie Monte Carlo n'est pas encore complètement comprise. En effet, de nombreuses discussions autour de ce sujet ont été publiés sur la mailing list computer-go, mais personne n'est encore parvenu à proposer un critère pour déterminer, sans expérimentations coûteuses de l'algorithme de MCTS, ce qu'est un bon simulateur de Monte Carlo. Ainsi, utiliser comme simulateur Monte-Carlo un bon joueur de Go, tel qu'un programme basé sur des règles expertes, peut réduire fortement les performances globales du MCTS, même à nombre de simulations fixé, i.e. indépendamment du surcoût en temps de calcul engendré. Une explication intuitive de ce phénomène pourrait être qu'un simulateur de Monte Carlo fortement biaisé mène à une estimation incorrecte de l'espérance de victoire pour les noeuds. Deuxièmement, la partie Bandit est encore trouble. Des implémentations utilisent encore la formule UCB (Auer *et al.*, 2002) mais avec une petite constante d'exploration, ainsi les coups avec le meilleur résultats empiriques sont utilisés, indépendamment du nombre de simulations, jusqu'à ce que la probabilité de victoire deviennent plus petite que la probabilité de victoire des noeuds non parcourus. Nous allons présenter à présent (i) les améliorations pour la partie "arbre" du MCTS (i.e. partie des simulations dans l'arbre) et (ii) les améliorations de la partie Monte-Carlo (i.e. partie des simulations hors de l'arbre).

Algorithm 3 Algorithme permettant de choisir un coup dans l'arbre, pour un jeu à somme nulle avec une récompense binaire (l'extension pour le cas de distributions arbitraires est directe). $Sims(s,d)$ est le nombre de simulations commençant à s avec comme premier coup d. Le nombre total de simulations pour une situation s est $Sims(s) = \sum_d Sims(s,d)$.

Fonction $decision = Bandit(situation\ s\ in\ the\ tree)$.
for d dans un ensemble de décisions possibles **do**
Soit $\hat{p}(d) = Wins(s,d)/Sims(s,d)$.
switch (formule de bandit) :

- **UCB1** : calculer $score(d) = \hat{p}(d) + \sqrt{2\log(Sims(s))/Sims(s,d)}$.
- **UCB-Tuned.1** : calculer $score(d) = \hat{p}(d) + \sqrt{\hat{V}\log(Sims(s))/Sims(s,d)}$ avec $\hat{V} = \max(0.001, \hat{p}(d)(1-\hat{p}(d)))$.
- **UCB-Tuned.2** : calculer $score(d) = \hat{p}(d) + \sqrt{\hat{V}\log(Sims(s))/Sims(s,d)} + \log(Sims(s))/Sims(s,d)$ avec $\hat{V} = \max(0.001, \hat{p}(d)(1-\hat{p}(d)))$.
- **exploration guidée par AMAF** : Calculer

$$score(d) = \alpha(d)\hat{p}(d) + (1-\alpha(d))\hat{\hat{p}}(d) \quad [1]$$

avec :

- $\hat{\hat{p}}(d)$ la proportion de simulations gagnées parmis les simulations AMAF utilisant la décision d dans la situation s (voir section 3) ;
- $\alpha(d)$ un coefficient dépendant de $Sims(s,d)$ (voir (Gelly *et al.*, 2007)).

end switch
end for
Retourner $\arg\max_d\ score(d)$.

(Coulom, 2006) et (Chaslot *et al.*, 2007) ont déjà combiné apprentissage hors-ligne (statistiques de parties professionnelles) et apprentissage en ligne (choix "bandit" de coups). (Gelly *et al.*, 2007) combine apprentissage en ligne (bandit) et transient (valeurs AMAF) et expérimente l'utilisation d'apprentissage hors-ligne, mais la partie hors-ligne (basée sur RLGO) a été ensuite supprimée de MoGo car l'amélioration était mineure et même négative après quelques améliorations de paramétrage. Nous présentons notre algorithme combinant :

– Apprentissage en ligne (bandit), en section 2 ;

– Apprentissage transient (valeur AMAF selon (Gelly *et al.*, 2007)), qui est assez similaire à l'agrégation d'état au sens où une simulation sur un état influence plusieurs états de manière transiente ; cette partie est présentée en section 3 ;

– Règles expertes, comme expliqué en 4 ;

– Apprentissage hors-ligne, grâce à des motifs selon (Chaslot *et al.*, 2007), comme expliqué en section 5 ;

– Préservation de diversité en simulations Monte-Carlo, cf section 6.1 ;

– Utilisation de contre-exemples, autour du Nakade comme expliqué en section 6.2.

La combinaison de toutes ces formes d'apprentissage est présentée en section 7.

2. Apprentissage en ligne : l'exploration Monte-Carlo d'arbre

Le principe de base du Monte-Carlo Tree Search est de construire, incrémentalement, un arbre des situations possibles ; la racine est le nœud courant, une arête est un coup légal, et une arête correspondant à un coup c relie une position à la position suivante lorsque le coup c est joué (i.e. l'arête relie la position avant le coup à la position après le coup). Un grand nombre de parties simulées est réalisé ; chaque simulation (i) met à jour des statistiques stockées dans les noeuds de l'arbre, (ii) augmente l'arbre en ajoutant des noeuds, et (iii) est elle-même influencée par les statistiques stockées précédemment dans l'arbre. La façon dont les statistiques stockées dans l'arbre influencent les simulations est le module "bandit".

L'algorithme de bandit est dédié à l'apprentissage en ligne de la valeur de certains coups ; il estime dynamiquement les coups à explorer. Dans beaucoup de programmes, ceci est fait par des algorithmes à la UCT, i.e. en choisissant le coup maximisant la quantité suivante :

$$score(move) = \underbrace{w(move)/n(move)}_{Exploitation} + C \underbrace{\sqrt{\log(\sum_{m \in M} n(m))/n(move)}}_{Exploration} \quad [2]$$

où :

– $n(move)$ est le nombre de fois où le coup a été exploré dans cette situation ;

– $w(move)$ est le nombre de fois où le coup a été essayé et a conduit à une victoire ;

– M est l'ensemble des coups possibles dans cette situation.

Une amélioration classique consiste à remplacer Eq. 2 par certaines versions améliorées (cf Algorithme 3), typiquement UCB-Tuned (Auer *et al.*, 2002), c'est une solution générale qui n'est pas spécifique aux jeux et qui peut être appliquée à la planification. Dans le cas des jeux, des formules comme les valeurs AMAF dans l'algorithme 3 peuvent être utilisées pour biaiser le terme d'exploration vers les coups qui paraissent intéressants selon les statistiques "AMAF" (Gelly *et al.*, 2007). Nous voulons introduire ici un biais par dessus ce biais en ajoutant des simulations AMAF virtuelles : ceci fera l'objet des parties 4 (ajout de règles expertes) et 5 (motifs).

3. Apprentissage transient : estimation rapide de la fonction valeur par heuristique "All Moves As First"

Depuis (Gelly *et al.*, 2007), MoGo utilise une exploration guidée par valeurs AMAF ; en partant de connaissances nulles, des statistiques AMAF sont collectées durant les simulations, ainsi que les statistiques usuelles de parties gagnées/perdues. Le principe est comme suit :

– Pour chaque simulation s avec récompense $r \in \{0, 1\}$, archivons la suite de coups $m_1, \ldots, m_n$ de la racine de l'arbre à la fin de la simulation ;

– Pour chaque nœud N de la simulation s avec récompense r, considérons les coups $m_k, m_{k+2}, m_{k+4}, \ldots, m_{k+2g}$ de la même couleur que N (même joueur au trait), jusqu'à la fin de la simulation ;

– Au noeud N, pour chaque i dans $\{1, 2, \ldots, g\}$, considérons ce qu'on appelle la simulation AMAF $m_i, m_{k+2}, m_{k+4}, \ldots, m_{k+2i-2}, m_k, m_{k+2i+2}, m_{k+2i+4}, \ldots, m_{k+2g}$ avec récompense r aussi.

Bien sûr, ces simulations AMAF sont 'fausses' ; on ne peut pas savoir si ces simulations sont consistantes (les règles ne sont pas forcément respectées), et la récompense est incertaine. Toutefois, la force de cette heuristique réside dans le fait que les statistiques sont collectées beaucoup plus vite : pour chaque simulation, on gagne un grand nombre de simulations dites AMAF - dans le cas du Go, environ la moitié du nombre d'intersections vides sur le goban. Ainsi, les valeurs AMAF sont biaisées, mais elles sont moyennées sur de plus grands échantillons - en conséquence, elles sont intrinsèquement transientes :

– au début de l'exploration d'un nœud, il n'y a pas de valeur AMAF ;

– plus tard, les statistiques standards sont très faibles car il y a peu de simulations, mais les valeurs AMAFs sont disponibles ; ainsi, le poids des valeurs AMAF est proche de 1 ;

– plus tard, les valeurs AMAF deviennent obsolètes car les statistiques standards s'affinent ; le poids des valeurs AMAF doit tendre vers 0.

Une faiblesse reste, due à l'initialisation : au tout début, ni les valeurs AMAF ni les statistiques standards ne sont disponibles ; on a besoin de connaissances a priori. De telles connaissances a priori sont des points forts dans e.g. CrazyStone ou Mango (Coulom, 2006; Chaslot *et al.*, 2007), mais dans ces programmes il n'y a pas de valeurs RAVE fournissant des informations transientes. Dans ce travail nous combinons avec succès (i) des connaissances expertes (section 4) (ii) des motifs extraits de bases de données comme dans Mango (section 5).

4. Apprentissage hors-ligne : ajout de connaissances expertes

La suppression de l'arbre des coups vraissemblablement mauvais a été expérimen-

tée ; cependant, de telles suppressions sont dangereuses au Go, car les exceptions aux règles sont fréquentes. Nous nous contenterons donc d'introduire un biais en faveur des coups "intéressants" en ajoutant des simulations virtuelles : au lieu de créer de nouveaux noeuds dans l'arbre sans aucune simulation, nous introduisons artificiellement des gains pour des coups qui sont supposés être bons ou, plus précisément, pour des coups qui sont supposés valoir la peine d'être explorés. Inversement, nous introduirons des pertes pour les coups supposés faibles. D'autres solutions pour inclure des connaissances expertes existent :

– Progressive widening (Coulom, 2007) ou progressive unpruning (Chaslot *et al.*, 2007) : le coup joué est choisi selon l'équation 2 *parmi les* n *coups préférés par l'heuristique*, n dépendant du nombre de simulations réalisés ;

– Progressive bias (Chaslot *et al.*, 2007) : la formule 2 est modifiée par un terme $+heuristic\,value(move)/n(move)$, dépendant de motifs ;

– First play urgency (Wang *et al.*, 2007) : le score représenté par la formule 2 est utilisé pour les coups m tels que $n(m) > 0$, et une valeur à priori dépendant des connaissances expertes est utilisée pour les autres coups ;

– MoGo incorpore un mélange de "Progressive bias" et de "Progressive widening" par l'ajout d'un bonus diminuant avec le nombre de simulations pour des coups proches au dernier coup joué. La distance est conçue à travers l'expertise Go : c'est une distance topologique qui suppose la distance 1 entre toutes les pierres d'une chaîne, quelle que puisse être la taille de cette chaîne.

Dans tous ces cas, l'impact des connaissances hors-lignes va en décroissant à mesure que les statistiques en ligne sont collectées. Les connaissances hors-ligne sont donc par nature différentes des connaissances transientes présentées en section 3 : alors que les connaissances transientes sont établies en ligne mais ont peu à peu un poids décroissant, les connaissances hors-lignes vont toujours decrescendo.

Divers éléments sur des connaissances expertes déjà publiés (e.g. (Coulom, 2007; Bouzy *et al.*, 2005)) incluent :

– les formes : dans (Coulom, 2007; Bouzy *et al.*, 2005), les formes sont générées automatiquement à partir d'un ensemble de données. Nous utilisons cette approche en section 5 ;

– les coups de capture (en particulier, le coup voisin à une nouvelle chaîne en atari), extension (notamment le shisho), l'évitement de se mettre en atari, l'atari (en particulier quand il y a un ko), la distance au bord (distance optimale $= 3$ en 19x19 Go), une petite distance au coup précédemment joué, une petite distance à l'avant-dernier coup, et aussi la caractéristique reflétant le fait que la probabilité (selon les simulations de Monte-Carlo) d'un emplacement à être de sa propre couleur à la fin de la partie est $\simeq 1/3$.

Les outils suivants sont utilisés dans nos implémentations en 19x19, et améliorent les résultats de l'heuristique AMAF :

– La ligne de territoire (i.e. ligne numéro 3) : 6.67 victoires sont ajoutées dans les simulations AMAF ;

– La ligne de la mort (i.e. première ligne) : 6.67 défaites sont ajoutées dans les simulations AMAF ;

– Peep-connect (ie. connecter deux chaînes quand l'adversaire menace de couper) : 5 gains sont ajoutés dans les simulations AMAF ;

– Hane (un coup qui "contourne" une ou plusieurs pierres de l'adversaire) : 5 gains sont ajoutés dans les simulations AMAF ;

– Connect : 5 victoires sont ajoutées dans les simulations AMAF ;

– Wall : 3.33 victoires sont ajoutées dans les simulations AMAF ;

– Bad Kogeima(même motif que le mouvement du cavalier aux échecs) : 2.5 défaites dans les simulations AMAF sont ajoutés ;

– Triangle vide (trois pierres de même couleur formant un triangle sans pierre adverse dans l'angle formé) : 5 défaites dans les simulations AMAF est ajoutée.

Ces formes sont illustrées sur la Figure 1. Avec un réglage naïf des paramètres à la main, elles fournissent 63.9 ± 0.5 % de victoires contre la version sans ces améliorations. Nous sommes optimistes sur le fait que le réglage des paramètres améliorera considérablement les résultats. D'ailleurs, dans les premiers développements de MoGo, certains bonus de "coupe" sont déjà inclus (i.e., les avantages de jouer à des emplacements qui vérifie des motifs de "coupe", i.e. des motifs pour lesquels un emplacement empêche l'adversaire de connecter deux groupes).

5. Apprentissage hors-ligne : data-mining hors-ligne de motifs

A l'instar de (Bouzy *et al.*, 2005), nous construisons un modèle pour évaluer la probabilité pour un coup d'être joué, si la partie était joué par un professionnel, conditionnellement au fait qu'il correspond à un certain motif. Cette probabilité, estimée sur 2000 parties professionnelles et comprenant des motifs de taille comprise entre 2 et 361 intersections, multipliée par un coefficient calibré empiriquement, est un bonus ajouté au score pour l'algorithme de bandit (cf Eq. 3 plus loin). La reconnaissance de motif est computationnellement chère et les premières expérimentations ont ainsi été négatives ; néanmoins, après calibrage des paramètres, nous avons obtenu des résultats clairement positifs.

En particulier, inclure ces modifications influence les paramètres suivants :

1) l'importance des statistiques en ligne augmente plus lentement (le coefficient α dans l'équation 3 augmente plus lentement) ;

2) le nombre de simulations d'un coup sortant de l'arbre avant création du noeud subséquent ; le coût de l'appel à l'heuristique est plus élevé que lorsque seuls les expertise de la section ci-dessus 4 étaient présents, car le coût de l'heuristique est nettement augmenté ;

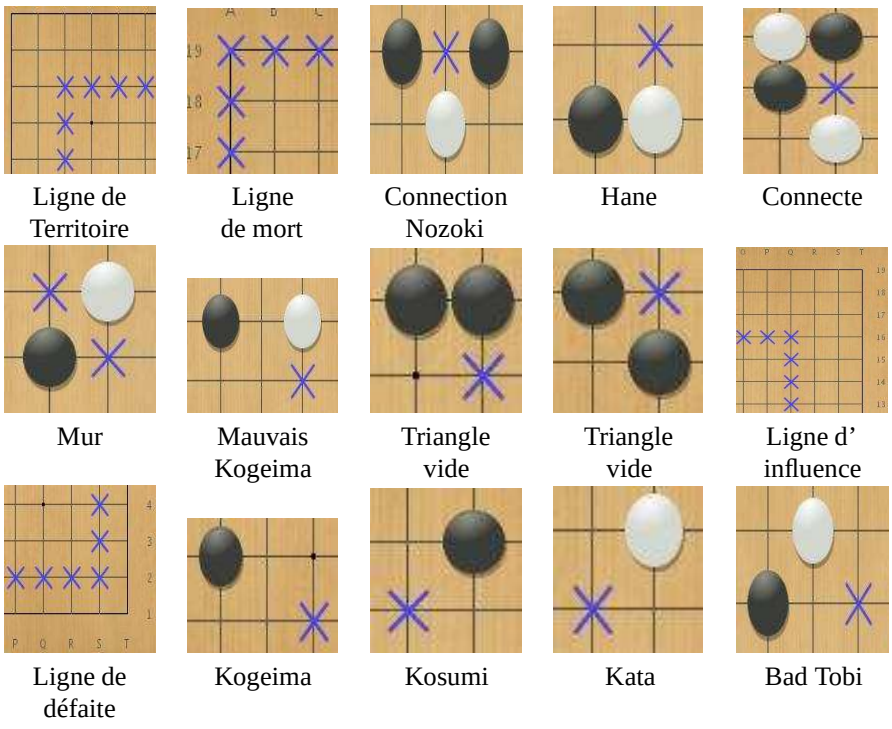

Figure 1. *Nous présentons ici les motifs exacts pour lesquels s'appliquent les bonus /malus. Dans tous les cas, les motifs sont présentés avec trait aux Noirs : les formes s'appliquent pour un coup Noir à une des croix. Les motifs avec couleurs inversées s'appliquent bien sûr aux Blancs. Dans le cas du Bad Kogeima, il ne doit pas y avoir de pierres noires autour de la croix pour que le bonus s'applique. Dans le cas du Kosumi et du Kogeima, il ne doit pas y avoir de pierres blanches entre la pierre noire et la croix. Le motif "seul sur la première ligne" s'applique seulement si les 5 emplacements autour du coup considéré sont vides. Dans le cas du bad tobi, il ne doit pas y avoir de pierre noire parmi les 8 emplacements autour de la croix. Dans le cas du Hane, la pierre blanche ne doit avoir aucune pierre amie parmi les 4 emplacements voisins.*

3) les coefficients optimaux des règles expertes de la section 4 sont modifiés.

Les résultats sont présentés en figure 2.

6. Amélioration des simulations Monte-Carlo (MC)

On ne sait pas encore comment créer un bon simulateur Monte-Carlo pour MCTS dans le cas du Go. De nombreuses personnes ont essayé d'améliorer le simulateur en

Version testée	Adversaire	Conditions des matchs	Taux de victoires
MoGo + motifs	Version sans motifs	3000 sims par coup	56 % $\pm$ 1.0 %
MoGo + motifs	Version sans motifs	2s par coup	50.9 % $\pm$ 1.5 %
MoGo + motifs + recalage/paramètres	MoGo + motifs	2s par coup	55.2 % $\pm$ 0.8 %

Figure 2. *Ajout de patterns extraites de parties professionnelles dans MoGo. Le premier recalage des paramètres est le recalage (i) des coefficients de compromis entre les différents apprentissages (ii) des paramètres de l'expertise Go (présentée en section 4 ; ces coefficients interfèrent clairement avec les motifs et sont donc naturellement à rééquilibrer) (iii) du nombre de simulations dans un coup avant de créer le noeud subséquent.*

augmentant sa force en tant que joueur indépendant, mais il a clairement été prouvé dans (Gelly *et al.*, 2007) que ce n'est pas un bon critère : un simulateur MC_1 qui joue beaucoup mieux qu'un simulateur MC_2 peux conduire à de très mauvais résultats une fois utilisé avec MCTS et ceci même en comparant à nombre de simulations identique. Certains simulateurs ont été appris à partir de base de données (Coulom, 2007), mais les résultats sont fortement améliorés par un réglage manuel des paramètres. Les raisons expliquant qu'un simulateur MC va être efficace étant encore inconnues, il est nécessaire d'expérimenter toute modification de manière intensive afin de la valider.

De nombreuses formes sont définies dans (Bouzy, 2005; Wang *et al.*, 2007; Ralaivola *et al.*, 2005). (Wang *et al.*, 2007) utilise des motifs et des connaissances expertes comme expliqué dans l'algorithme 2. Nous présentons ci-dessous deux nouvelles améliorations, toutes deux centrées sur l'augmentation de la diversité des simulations : l'impact des deux modifications augmente avec le nombre de simulations.

6.1. *"Fill board" : diversifier les simulations*

Le principe de cette modification est de jouer en priorité aux emplacements du plateau situés au centre de zones vides (où il n'y a pas encore de pierre). Le but est de diversifier les simulations en jouant de maniere moins locale. En effet, le simulateur MC de MoGo utilise un grand nombre de motifs de taille 3*3. Quand un ou plusieurs motif est trouvé sur le plateau, alors le coup correspondant est joué. La taille de ces motifs implique que le coup joué sera très proche des pierres déja présentes. En particulier en début de partie, le nombre de pierres est faible et le nombre de coups possibles est alors très restreint (voir figure 3 (gauche)). Cette modification intervenant avant la détection de motifs, elle permet d'obtenir des simulations plus diversifiées.

Comme essayer chaque emplacement du plateau serait trop coûteux en temps, nous utilisons la procédure suivante : une position du plateau est choisie au hasard ; si les 8

emplacements contiguës sont vides alors le coup est joué, sinon on essaye les $N-1$ positions suivantes du plateau ; N est un paramètre de la modification "fill board". Un N plus grand implique un plus grand coût de calcul et une plus grande diversité.

L'algorithme détaillé est présenté dans l'algorithme 4.

Algorithm 4 Algorithme permettant de choisir un coup dans les simulations MC, comprenant l'amélioration "fill board". Nous avons également expérimenté avec une contrainte de 4 et 24 emplacements libres au lieu de 8 mais les résultats furent décevants.

if le dernier coup est un atari **then**
  Sauver les pierres en atari.
else
  partie "Fill board".
  for $i \in \{1, 2, 3, 4, \ldots, N\}$ **do**
    Tirer une position x au hasard sur le plateau.
    Si x est un emplacement vide et que les 8 emplacements autour sont également vide, jouer x (sortir).
  end for
  Fin de la partie "fill board".
  if il y a un emplacement vide parmi les 8 emplacements autour du dernier coup qui correspond à un pattern **then**
    Jouer au hasard sur l'un de ces emplacements (sortir).
  else
    if il y a un coup qui capture des pierres **then**
      Capturer les pierres (sortir).
    else
      if il reste des coups légaux **then**
        Jouer au hasard un coup légal (sortir).
      else
        Renvoyer "passe".
      end if
    end if
  end if
end if

Les expériences sont présentées dans la figure 3 (droite).

6.2. *Le problème du "Nakade"*

. Une faiblesse connue de MoGo, ainsi que de beaucoup de programme MCTS, est que les "nakade" ne sont pas gérés correctement. Nous utiliserons le terme *nakade* pour désigner une situation dans laquelle un groupe encerclé possède un unique espace interne enfermé à partir duquel le joueur ne pourra pas créer deux yeux si l'adversaire répond correctement. Le groupe est donc mort mais le simulateur MC utilisé dans la

Jeu 9x9		Jeu 19x19	
Nb de sims par coup ou temps par coup	Taux de succès	Nb de sims par coup ou temps par coup	Taux de succès
10 000	52.9 % ± 0.5%	10000	49.3 ± 1.2 %
5s/coup, 8 coeurs	54.3 % ± 1.2 %	5s/coup, 8 coeurs	77.0 % ± 3.3 %
100 000	55.2 % ± 1.4 %	100 000	73.7 % ± 2.9%
200 000	55.0 % ± 1.1 %	200 000	78.4 % ± 2.9 %

Figure 3. *Gauche : perte de diversité quand on n'utilise pas l'option "fill board" : la pierre blanche est la dernière pierre jouée, une simulation MC pour le joueur noir commencera obligatoirement par l'une des cases marquée d'un triangle. Droite : Résultats associés à la modification "fill board". Comme la modification induit un surcoût computationnel, les résultats sont meilleurs pour un nombre de simulations par coup fixé ; cependant, l'amélioration est clairement significative. Le surcoût computationnel est réduit lorsque l'on utilise plusieurs cœurs : en effet, il y a moins de concurrence pour l'accès mémoire lorsque les simulations sont plus longues. C'est pourquoi la différence en temps entre des simulations simples et complexes se réduit avec le nombre de cœurs. Cet élément montre l'intérêt accru des simulations complexes dans le cadre de la parallélisation : même si des simulations coûtent plus cher, ce surcoût est "épongé" par la parallélisation.*

version de MoGo de référence (Algorithme 2) va estimer que le groupe vit avec une forte probabilité. En effet, le simulateur ne répondra pas les coups considérés comme évidents même par un joueur moyen et peut donc conduire à la survie du groupe. L'arbre va donc sous estimer le danger correspondant à cette situation et éventuellement se développer dans cette direction. La partie MC doit donc être modifiée pour que la probabilité de gagner prenne en compte ces situations de *nakade* .

Il est intéressant de noter que MoGo étant principalement développé en jouant contre lui-même, cette faiblesse n'est apparue qu'une fois trouvée par un humain (voir le mail de D.Fotland "UCT and solving life and death" sur la mailing list computer-go).

Il serait théoriquement possible d'encoder dans les simulations MC un vaste ensemble de comportements relatifs au *nakade* , mais cette approche a deux faiblesses : (i) cela serait coûteux en temps et les simulations doivent être rapides (ii) changer le comportement du simulateur MC de manière trop abrupte conduit généralement à des résultats décevants. C'est pourquoi la modification a été créée ainsi : si on trouve un ensemble d'exactement 3 intersections libres entourées de pierres adverses, on joue au centre (le point vital). Cela permet de fortement se rapprocher de la vraie probabilité de gagner dans la plupart des situations de *nakade* . Le nouvel algorithme est présenté dans l'algorithme 5.

Algorithm 5 Nouveau simulateur MC, permettant d'éviter le problème de *nakade* .

if le dernier coup est un atari **then**
  Sauver les pierres en atari.
else
  Début de la modification *nakade*
  for x parmis l'un des 4 emplacements libres autour du dernier coup joué **do**
    if x fait parti d'un trou de 3 emplacements contigus entourés de pierres adverses **then**
      Jouer au centre du trou (sortir).
    end if
  end for
  Fin de la modification *nakade*
  partie "Fillboard".
  for $i \in \{1, 2, 3, 4, \ldots, N\}$ **do**
    Tirer une position x au hasard sur le plateau.
    Si x est un emplacement vide et que les 8 emplacements autour sont également vide, jouer x (sortir).
  end for
  Fin de la partie "fill board".
  if il y a un emplacement vide parmi les 8 emplacements autour du dernier coup qui correspond à un pattern **then**
    Jouer au hasard sur l'un de ces emplacements (sortir).
  else
    if il y a un coup qui capture des pierres **then**
      Capturer les pierres (sortir).
    else
      if il reste des coups légaux **then**
        Jouer au hasard un coup légal (sortir).
      else
        Renvoyer "passe".
      end if
    end if
  end if
end if

Cette approche est validée par deux expériences différentes : (i) des positions connues où la version de MoGo de référence ne choisit pas le bon coup (figure 4) (ii) des parties nouveau MoGo contre MoGo de référence (table 1).

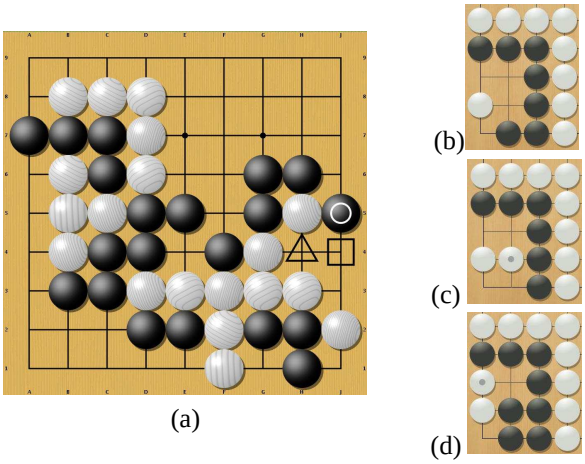

Figure 4. *figure (a) (une partie réelle jouée et perdue par MoGo), MoGo (blanc) sans la modification pour les* nakade *joue H4 ; noir répond J4 et le groupe F1 est mort (MoGo perd). Après la modification, MoGo joue J4 qui est le bon coup car il permet de vivre après un ko. Les exemples (b), (c) et (d) sont des situations similaires dans lesquelles MoGo ne réalise pas que son groupe est mort. Dans chacun des cas, la modification permet de résoudre le problème.*

Nb de simulations par coup	Taux de succès	Nb de simulations par coup	Taux de succès
plateau 9x9		plateau 19x19	
10000	52.8 % $\pm$ 0.5%		
100000	55.6 % $\pm$ 0.6 %	100 000	53.2 % $\pm$ 1.1%
300000	56.2 % $\pm$ 0.9 %		
5s/coup, 8 coeurs	55.8 % $\pm$ 1.4 %		
15s/coup, 8 coeurs	60.5 % $\pm$ 1.9 %		
45s/coup, 8 coeurs	66.2 % $\pm$ 1.4 %		

Tableau 1. *Validation expérimentale de la modification* nakade *: MoGo modifié contre MoGo de référence. Il apparaît que plus le nombre de simulations est important (ce qui est directement lié au niveau), plus l'impact de la modification est important.*

7. Combiner hors-ligne, transient et en ligne

Dans cette section on présente la combinaison d'apprentissage en ligne (module bandit, section 2), apprentissage transient (valeurs AMAF/RAVE, section 3), connaissances expertes (section 4) et motifs extraits hors-ligne (section 5).

Nous soulignons que cette combinaison est loin d'être directe : en raison de l'équilibre subtil entre apprentissage en ligne (bandit) et apprentissage transient (RAVE), les premières expériences étaient très négatives ; les résultats n'ont commencé à apparaitre qu'après calibrage fin des paramètres.

Le score étend l'exploration AMAF-guidée (Eq. 1) comme suit :

$$Score(d) = \underbrace{\alpha \hat{p}(d)}_{Valeur\ en\ ligne} + \underbrace{\beta \hat{\hat{p}}(move)}_{Valeur\ transiente} + \underbrace{\gamma H(d)}_{Valeur\ hors-ligne} + \delta P(d,n) \quad [3]$$

où les coefficients α, β, γ et δ sont calibrés empiriquement en fonction de $n(d)$ (nombre de simulations de la décision d) et n (nombre de simulations du noeud courant) avec les contraintes qui suivent :

– $\alpha + \beta + \gamma = 1$;

– $\alpha \simeq 0$ et $\beta \simeq 0$, i.e. $\gamma \simeq 1$, pour $n(d) = 0$;

– $\beta >> \alpha$ pour $n(d)$ petit ;

– $\alpha \simeq 1$ pour $n(d) \to \infty$;

– δ dépend seulement de n et décroit vers 0 comme $n \to \infty$;

– $d \mapsto P(d,n)$ converge vers la fonction constante comme $n \to \infty$; $\delta P(d,n)$ est analogue au terme dit "désélagage progressif" ((Chaslot *et al.*, 2007)).

Ces règles impliquent que :

– initialement, le poids le plus fort est pour le hors-ligne ;

– plus loin, la partie la plus importante est l'apprentissage transient (valeurs RAVE) ;

– ultimement, seules les "vraies" statistiques sont importantes.

Les coefficients sont modifiés à chacune des fréquentes amélioration des valeurs hors-ligne et la part correspondante du code peut être fournie sur demande.

8. Conclusion

Nos conclusions sont les suivantes :

– Comme pour les humains, toutes les échelles de temps sont importantes :

- apprentissage hors-ligne (règles stratégiques et motifs) comme dans (Coulom, 2007; Chaslot *et al.*, 2007) ;

- apprentissage en ligne (i.e. analyse de séquences par simulation) (Chaslot *et al.*, 2006; Coulom, 2006)

- information transiente (extrapolation comme guide pour l'exploration).

– Réduire délicatement la diversité des simulations Monte-Carlo a longtemps été une bonne idée en exploration Monte-Carlo d'arbres ; (Wang *et al.*, 2007) a montré qu'introduire différents motifs dans le simulateur Monte-Carlo augmentait considérablement l'efficacité. Toutefois, de nombreuses expérimentations autour de l'augmentation du niveau du simulateur Monte-Carlo en tant que joueur proprement dit a donné des résultats négatifs quant au niveau de jeu du joueur MCTS construit sur ce simulateur - augmenter le niveau de jeu du simulateur MC n'augmente pas le niveau de jeu du MCTS. Le fait même que le bon compromis entre niveau de jeu du MCTS et diversité soit la solution n'est pas clair. On peut seulement affirmer clairement qu'augmenter la diversité devient de plus en plus important à mesure que la puissance de calcul augmente, comme montré dans la section 6.

– Alors que l'exploration est important lorsque l'apprentissage pour le guider est insuffisant, les constantes optimales dans le terme d'exploration deviennent 0 (cf Eqs. 2 et 3) quand l'apprentissage est amélioré. Dans MoGo, la constante devant le terme d'exploration était > 0 avant l'introduction de valeurs RAVE dans (Gelly *et al.*, 2007) ; elle est désormais nulle - on renonce à un coup une fois que l'on a plutôt une mauvaise estimation de sa valeur, et pas simplement par diversification vers d'autres coups.

Remerciements

Nous remercions Bruno Bouzy, Rémi Munos, Yizao Wang, Rémi Coulom, Tristan Cazenave, Jean-Yves Audibert, David Silver, Martin Mueller,KGS, Cgos, ainsi que toute la mailing list computer-go pour toutes les discussions intéressantes que nous avons pu avoir. Un grand merci à la fédération française de Go et à Recitsproque pour avoir organisé un match officiel contre un humain de haut niveau ; un grand merci également à tous les joueurs de la fédération française de Go qui ont accepté de jouer et de commenter des parties test contre MoGo, et à Catalin Taranu, joueur professionnel 5e dan, pour ses commentaires après le challenge IAGO.

9. Bibliographie

Agrawal R., « The Continuum-Armed Bandit Problem », *SIAM J. Control Optim.*, vol. 33, n° 6, p. 1926-1951, 1995.

Auer P., Cesa-Bianchi N., Fischer P., « Finite-time analysis of the multiarmed bandit problem », *Machine Learning*, vol. 47, n° 2/3, p. 235-256, 2002.

Banks J. S., Sundaram R. K., « Denumerable-Armed Bandits », *Econometrica*, vol. 60, n° 5, p. 1071-96, September, 1992. available at http ://ideas.repec.org/a/ecm/emetrp/v60y1992i5p1071-96.html.

Bouzy B., « Associating domain-dependent knowledge and Monte Carlo approaches within a go program », *in* K. Chen (ed.), *Information Sciences, Heuristic Search and Computer Game Playing IV*, vol. 175, p. 247-257, 2005.

Bouzy B., Chaslot G., « Bayesian generation and integration of k-nearest-neighbor patterns for 19x19 Go », *G. Kendall and Simon Lucas, editors, IEEE 2005 Symposium on Computational Intelligence in Games, Colchester, UK*, p. 176-181, 2005.

Chaslot G., Saito J.-T., Bouzy B., Uiterwijk J. W. H. M., van den Herik H. J., « Monte-Carlo Strategies for Computer Go », *in* P.-Y. Schobbens, W. Vanhoof, G. Schwanen (eds), *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, p. 83-91, 2006.

Chaslot G., Winands M., Uiterwijk J., van den Herik H., Bouzy B., « Progressive Strategies for Monte-Carlo Tree Search », *in* P. Wang et al. (eds), *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, World Scientific Publishing Co. Pte. Ltd., p. 655-661, 2007.

Coquelin P.-A., Munos R., « Bandit Algorithms for Tree Search », *Proceedings of UAI'07*, 2007.

Coulom R., « Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search », *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, 2006.

Coulom R., « Computing Elo Ratings of Move Patterns in the Game of Go », *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.

Dani V., Hayes T. P., « Robbing the bandit : less regret in online geometric optimization against an adaptive adversary », *SODA '06 : Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, ACM Press, New York, NY, USA, p. 937-943, 2006.

Gelly S., Silver D., « Combining online and offline knowledge in UCT », *ICML '07 : Proceedings of the 24th international conference on Machine learning*, ACM Press, New York, NY, USA, p. 273-280, 2007.

Kocsis L., Szepesvari C., « Bandit-based Monte-Carlo Planning », *ECML'06*, p. 282-293, 2006.

Lai T., Robbins H., « Asymptotically efficient adaptive allocation rules », *Advances in applied mathematics*, vol. 6, p. 4-22, 1985.

Ralaivola L., Wu L., Baldi P., « SVM and pattern-enriched common fate graphs for the game of Go », *Proceedings of ESANN 2005*, p. 485-490, 2005.

Wang Y., Gelly S., « Modifications of UCT and sequence-like simulations for Monte-Carlo Go », *IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii*, p. 175-182, 2007.

Article reçu le 22/09/1996.
Version révisée le 04/10/2005.
Rédacteur responsable : GUILLAUME LAURENT

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com

ANNEXE POUR LE SERVICE FABRICATION
A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :
   RIA. Volume X – n°Y/Z

2. AUTEURS :
   Louis Chatriot— Christophe Fiter— Guillaume Chaslot— Sylvain Gelly— Jean-Baptiste Hoock— Julien Perez— Arpad Rimmel— Olivier Teytaud

3. TITRE DE L'ARTICLE :
   Combiner connaissances expertes, hors-ligne, transientes et en ligne pour l'exploration Monte-Carlo

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   Apprentissage et MC

5. DATE DE CETTE VERSION :
   11 juillet 2008

6. COORDONNÉES DES AUTEURS :

   – adresse postale :
   TAO (Inria), LRI, UMR 8623 (CNRS - Univ. Paris-Sud),
   bat 490 Univ. Paris-Sud 91405 Orsay, France, teytaud@lri.fr
   – téléphone : 00 00 00 00 00
   – télécopie : 00 00 00 00 00
   – e-mail : publieur

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LaTeX, avec le fichier de style `article-hermes.cls`,
   version 1.23 du 17/11/2005.

8. FORMULAIRE DE COPYRIGHT :
   Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
   `http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com